PROCESSING SYSTEM AND METHOD INCLUDING A DEDICATED COLLECTIVE OFFLOAD ENGINE PROVIDING COLLECTIVE PROCESSING IN A DISTRIBUTED COMPUTING ENVIRONMENT

Field of the Invention

[0001] This invention relates in general to a distributed computing environment, and in particular, to a processing system and method including a dedicated collective offload engine providing collective processing of distributed data received from processing nodes in a distributed computing environment.

Background of the Invention

[0002] Collective processing is the collective combination and dissemination of information across processes in a distributed computing environment. Performance of conventional collective processes on large distributed computing environments is a key determinant of system scalability.

[0003] Typically, implementation of collective processing includes using a software tree approach, where message passing facilities are used to form a virtual tree of processes. One drawback of this approach is the serialization of delays at each stage of the tree. These delays are additive in the overall overhead associated with the collective processing. Furthermore, this software tree approach results in a theoretical logarithmic scaling latency of the overall collective processing versus system size. Due to interference from daemons, interrupts and other background activity, cross traffic, and the unsynchronized nature of independent operating system images and their dispatch cycles, measured values of scaling latency are usually significantly worse than theoretical values.

[0004] Scaling latency has been minimally improved by enhancement attempts that include tuning communication protocol stacks and provision of certain schedulers. Techniques such as "daemon squashing" to synchronize daemon activity are likely to bring practical results in line with theoretical ones, but the cost of multiple traversals of the software tree-based protocol stack remain.

[0005] Accordingly, a need remains for a novel collective processing approach that, for example, mitigates the large latency associated with a software tree approach.

Summary of the Invention

[0006] The shortcomings of the prior art are overcome and additional advantages are provided through the provision of a processing system comprising a dedicated collective offload engine. The dedicated collective offload engine is, for example, coupled to a switch fabric of a distributed computing environment having multiple processing nodes also coupled to the switch fabric. Further, the dedicated collective offload engine, for instance, provides collective processing of data from at least some processing nodes of the multiple processing nodes and produces a result based thereon. The result is forwarded to at least one processing node of the multiple processing nodes.

[0007] A method, computer program products, and a data structure corresponding to the above-summarized system are also described and claimed herein.

[0008] Additional features and advantages are realized through the techniques of the present invention. Other embodiments and aspects of the invention are described in detail herein and are considered a part of the claimed invention.

Brief Description of the Drawings

[0009] The subject matter which is regarded as the invention is particularly pointed out and distinctly claimed in the claims at the conclusion of the specification. The foregoing and other objects, features, and advantages of the invention are apparent from the following detailed description taken in conjunction with the accompanying drawings in which:

[0010] FIG. 1 is a block diagram of one embodiment of a distributed computing environment incorporating and using one or more aspects of the present invention;

[0011] FIG. 2 is a block diagram of one embodiment of a dedicated collective offload engine, in accordance with one or more aspects of the present invention; and

[0012] FIG. 3 depicts one embodiment of a packet data structure to be sent by a processing node and used in collective processing by a dedicated collective offload engine, in accordance with one or more aspects of the present invention.

Best Mode for Carrying Out the Invention

[0013] In accordance with an aspect of the present invention, a facility is provided which enables high performance collective processing of data in a distributed computing environment. In particular, collective processing is performed by a dedicated collective offload engine in communication with processing nodes of the distributed computing environment across a switch fabric.

[0014] One embodiment of a distributed computing environment, generally denoted 100, incorporating and using one or more aspects of the present invention is depicted in FIG. 1. As shown, distributed computing environment 100 includes, for example, multiple processing nodes 102a – 102d coupled to a switch fabric 106 and at least one dedicated collective offload engine 104a – 104c also coupled to switch fabric 106. Dedicated collective offload engines 104a – 104c are, for example, coupled to switch fabric 106 in a manner similar to the coupling of processing nodes 102a – 102d to switch fabric 106. Communications are transmitted via switch fabric 106 between, for instance, two processing nodes 102a and 102b, a processing node and a dedicated collective offload engine 102a and 104a, or two (or more) dedicated collective offload engines 104a and 104b. Examples of switch fabric 106 include an InfiniBand switch and a Federation switch, both offered by International Business Machines Corporation of Armonk, New York. Multiple processing nodes 102 are, for example, servers, such as RS/6000 or eServer pSeries servers offered by International Business Machines Corporation. Multiple processing nodes 102 may be homogeneous or heterogeneous processing nodes.

[0015] In accordance with an aspect of the present invention, dedicated collective offload engine 104a is, for example, a dedicated hardware device built from, for instance, field programmable gate arrays (FPGAs). Collective offload engine 104a is a specialized device dedicated to providing collective processing of data from at least some of the processing nodes 102a – 102d. For example, collective offload engine 104a might receive data contributions from certain participating processing nodes of the multiple processing nodes connected to switch fabric 106. For example, processing nodes 102a – 102c might send data to be globally summed. Instead of using the conventional software tree approach whereby the data contributions are distributively summed at various levels of a tree across the network, the collective operation of global summing is offloaded to dedicated collective offload engine 104a. At the dedicated collective offload engine, collective processing is applied to the data contributions of the processing nodes to produce a global sum, and this result is then forwarded to one or more processing nodes, such as the data contributing nodes. Other collective operations, including standard Message Passing Interface (MPI) collective operations could be substituted for the global summing operation in this example. Operation of a dedicated collective offload engine is described in more detail below relative to FIG. 2.

[0016] The capacity of collective processing required in a distributed computing environment depends on contributed data payload lengths and system size, which is indicated by, for example, the number of processing nodes. To accommodate various collective processing capacities, various modifications can be made to the computing environment described above. One approach is to use cascaded multiple dedicated collective offload engines 104a – 104c that are coupled to switch fabric 106 and in communication with each other via the switch fabric. A second approach employs multiple dedicated collective offload engines attached to switch fabric 106 with a private channel between them, as depicted by the dashed bi-directional arrow between dedicated collective offload engines 104b and 104c. In this private channel approach, one dedicated collective offload engine 104b could combine, for example, results from odd numbered processing nodes while dedicated collective offload engine 104c might combine results from even numbered processing nodes. Using their private channel, the two results are then combined and known to each dedicated collective offload engine. Each collective offload engine could then broadcast the combined result to interested processing nodes. A third approach provides a multi-ported dedicated collective offload engine. Other approaches may include various combinations of the three approaches described above.

[0017] One embodiment of a dedicated collective offload engine, in accordance with an aspect of the present invention, is depicted in FIG. 2 and generally denoted 200. Within dedicated collective offload engine 200, an adapter, such as a host channel adapter 202, is coupled to and facilitates communication across switch fabric 106 (FIG. 1) using a link protocol. Host channel adapter 202 is, for example, a GigE, Federation or SP Switch2 adapter. Interface logic 204 provides an interface between host channel adapter 202 and a dispatcher 206 and a payload memory 208. Dispatcher 206 is implemented in, for example, FPGAs, and is the entity that executes protocol to process packets received from and sent to processing nodes 102a – 102d. Payload memory 208 receives and stores data contributions from processing nodes 102a – 102d participating in a collective operation. These data contributions are included in, for instance, vector operands of collective operations used in collective processing of the data contributions. A pipelined arithmetic logic unit (ALU) 210 retrieves and performs the collective processing of the data contributions. ALU 210 might operate at, for instance, approximately 250 Mflop/s (mega floating point operations per second). Dispatcher 206 controls collective processing of the data contributions by, for example, directing ALU 210 to the payload memory storage locations that contain particular data contributions required for a collective operation. In the global sum example described above, payload memory 208 would receive data contributed by participating processing nodes and dispatcher 206 would track where that data is located and direct ALU 210 to perform the summing of the data.

[0018] Collective processing control by dispatcher 206 also extends to tracking both process task information associated with the participating processing nodes and synchronization group information. Process task identification information is stored in task tables 212 and provides addressing information for packets directed to processing nodes 102 by dispatcher 206. Synchronization group identification information is stored in synchronization group tables 214 and allows dispatcher 206 to determine, for instance, when data contributions from the processing nodes in the synchronization group are received. Moreover, the identification information in synchronization group tables 214 allows dispatcher 206 to identify which synchronization group is associated with a particular data contribution, even when multiple synchronization groups correspond to the process task that generates the data contribution. This identification facilitates collective processing of data that is associated with, for example, a single synchronization group.

[0019] Synchronization groups are groups of processing nodes which are used to perform, for example, sub-computations in "divide and conquer" algorithms. Sub-computations are completed within each synchronization group and dispatcher 206 would recognize when the result from each sub-computation has been received by payload memory 208. Once all sub-computation results have been received, dispatcher 206 provides the storage locations of the sub-computation results to ALU 210 and directs the

ALU to perform collective processing using the sub-computation results. This collective processing produces a final result, which is then forwarded to the interested participating processing nodes. One example of a synchronization group is an MPI communicator or communicating group, which identifies a subset of processes (or tasks) of a parallel job to which a collective operation can be directed without affecting processes outside the identified subset.

[0020] FIG. 3 depicts one example of a data structure for a data packet 300 sent from a processing node participating in collective processing by a dedicated collective offload engine. This exemplary packet includes a standard packet header field 302, the contents of which depend on the switching technology implemented. With Ethernet switching, for instance, an Internet Protocol header would be used. A JobId field 304 identifies the currently running parallel application or job to which packet 300 pertains. A TaskId field 306 identifies the task within the identified job that sent packet 300. SynchGroupId field 308 identifies the synchronization group of processing nodes to which the operation identified by packet 300 pertains. MemberNum field 310 indicates which member of the synchronization group sent packet 300. OpCode field 312 specifies the type of collective operation to be performed. Examples of types of collective operations include floating point maximum, floating point sum, and integer sum. Payload field 314 includes the data contributed by the identified processing node for the specified collective operation. As noted, data in payload 314 is received and stored in payload memory 208 and retrieved by ALU 210 (FIG. 2) under the direction of dispatcher 206. ALU 210 performs the collective operation specified in OpCode 312 using the data in payload 314.

[0021] The following performance analysis example illustrates the performance advantage of a processing system based on a dedicated collective offload engine (COE), as described herein, when used for a barrier operation in a distributed computing environment using an available interconnect switch fabric.

[0022] Assumptions used in the performance analysis include:

Interconnect latency for MPI point-to-point = 5 μs
Interconnect link bandwidth = 2GB/s
Barrier packet size including headers = 64 bytes
Interconnect fabric latency = 0.8 μs
One COE serves approximately 32 nodes; 32 COEs serve a 1024-node system.

[0023] The steps to execute a barrier operation using COEs are:

1. Each processing node sends a 64-byte packet to its COE (MPI send side overhead) in less than 2.5 μs.
2. 32 processing nodes $\times$ 64 bytes each = 2KB arrives at the COE in 1 μs.
3. COE pipeline latency is less than 2 μs.
4. COE sends 2KB of reply data to the 32 processing nodes in 1 μs.
5. Processing nodes receive replies (MPI receive side overhead) in less than 2.5 μs.

[0024] Thus, the total time for the 32 processing nodes is approximately 9 μs. For a 1024-processing node system, the COEs can be cascaded into two stages resulting in cascading overhead of approximately 5 μs. Therefore, the total time for a barrier operation in a 1024-processing node system is approximately 14 μs.

[0025] In contrast, the theoretical performance for a 1024-node barrier operation using a conventional software tree approach is approximately $2 \log_2 1024 \times 5$ μs = 100 μs, assuming no overhead and delays due to interference from daemons, interrupts and other operating system activity. In this expression, the factor of 2 indicates that the software tree is traversed in both up and down directions, and $\log_2 1024$ is the height of the tree. The factor of 5 μs is the latency of a single point-to-point message.

[0026] Thus, in this case, the dedicated collective offload engine improves collective processing performance approximately by a factor of 7 versus the theoretical, optimistic estimate for the software tree approach.

[0027] Although the above example is based on an MPI collective operation, those skilled in the art will note that the dedicated collective offload engine is applicable to other operations to achieve enhanced system performance. For example, distributed lock management operations associated with distributed databases and distributed file systems can be implemented in a COE-based processing system where distributed locks are stored in the dedicated collective offload engine. As another example, in a cluster/parallel file system, global atomic operations and global system level locks can be implemented using the collective offload engine. Global synchronization primitives in, for instance, the Unified Parallel C programming language can also be stored in the collective offload engine. Further, a collective offload engine could be used as a mechanism to scatter an executable file or replicate data used by a parallel job to multiple processing nodes associated with the parallel job.

[0028] Those skilled in the art will note from the above discussion that one or more aspects of the present invention advantageously include a dedicated collective offload engine that provides collective processing of data from processing nodes in a distributed computing environment. This enables, for example, an approach that accelerates collective processing and promotes deterministic performance in a manner that enhances scalability of distributed computer systems. Thus, these beneficial characteristics of the dedicated collective offload engine decrease the average time required to perform collective operations and reduce random fluctuations in those time periods. Moreover, because the dedicated collective offload engine(s) is positioned apart from mainline communication paths between processing nodes via the fabric switch, it does not interfere with the performance or development of these mainline paths. Further, the dedicated collective offload engine is extensible because it can, for example, be easily adapted to various types of switch fabrics.

[0029] The present invention can be included in an article of manufacture (e.g., one or more computer program products) having, for instance, computer usable media. The media has embodied therein, for instance, computer readable program code means or logic (e.g., instructions, code, commands, etc.) to provide and facilitate the capabilities of the present invention. The article of manufacture can be included as a part of a computer system or sold separately.

[0030] Additionally, at least one program storage device readable by a machine, embodying at least one program of instructions executable by the machine to perform the capabilities of the present invention can be provided.

[0031] The flow diagrams depicted herein are just examples. There may be many variations to these diagrams or the steps (or operations) described therein without departing from the spirit of the invention. For instance, the steps may be performed in a differing order, or steps may be added, deleted or modified. All of these variations are considered a part of the claimed invention.

[0032] Although preferred embodiments have been depicted and described in detail herein, it will be apparent to those skilled in the relevant art that various modifications, additions, substitutions and the like can be made without departing from the spirit of the invention and these are therefore considered to be within the scope of the invention as defined in the following claims.